N89-16369

Ah!Help: A Generalized On-line Help Facility

Wong Nai Yu*
Charmiane Mantooth*
Alex Soulahakil*

1. Introduction

In modern-day programming it is not sufficient that a large, commercially available, software package simply work. It must have certain characteristics that make it marketable. Among these, user friendliness and portability are of major importance.

In the user friendliness category, a great forward step was taken with the introduction of on-line help facilities, relieving the user from continually turning to cumbersome manuals for assistance. The on-line help facility we have designed is neither unique nor revolutionary. It is a simple program which was originally designed to work in conjunction with a screen generation package. It is, however, independent of it and is, therefore, portable. The only new aspect introduced by this package is that it is written in Ada and utilizes certain Ada facilities, such as binary files and Direct_Io, which make implementation neater, simpler and more straight-forward than languages have heretofore allowed. In addition, the program uses only standard Ada generics, thus adding to its portability.

The concept behind this package is to allow the building of help files in textual format. The program then builds a binary file, creating and storing an index for the named file. This index, along with secondary indices created for further help on specific choices made available to the user, is later used to access the help file associated with the program currently being used by the user. Upon exiting the help mode, the user is returned to the point from where the on-line help was requested.

2. Design Considerations

The basic and overall purpose of this program was to produce a convenient, easy to use, general purpose on-line help facility. Convenience and ease of use from the end-user's point of view, however, usually means time consuming, difficult programming for the low level designer. It also generally means that there is a loss of generality and/or portability.

In order to combine these seemingly incongruent features we decided that an implementor/designer should be able to provide an on-line help menu and associated verbal descriptions in textual format, via a text file. The program should take over from there and create the actual manual which the user sees. The text file method does, however, present certain restrictions. It is highly formatted, meaning the designer of an on-line help manual must be familiar with the rules imposed by the program. We felt this to be only a minor inconvenience in comparison to the advantages to be gained by the program.

The main design considerations were that the user should, by use of a function key, be able to call an on-line help facility pertinent to whatever mode/program he or she happened to be in.

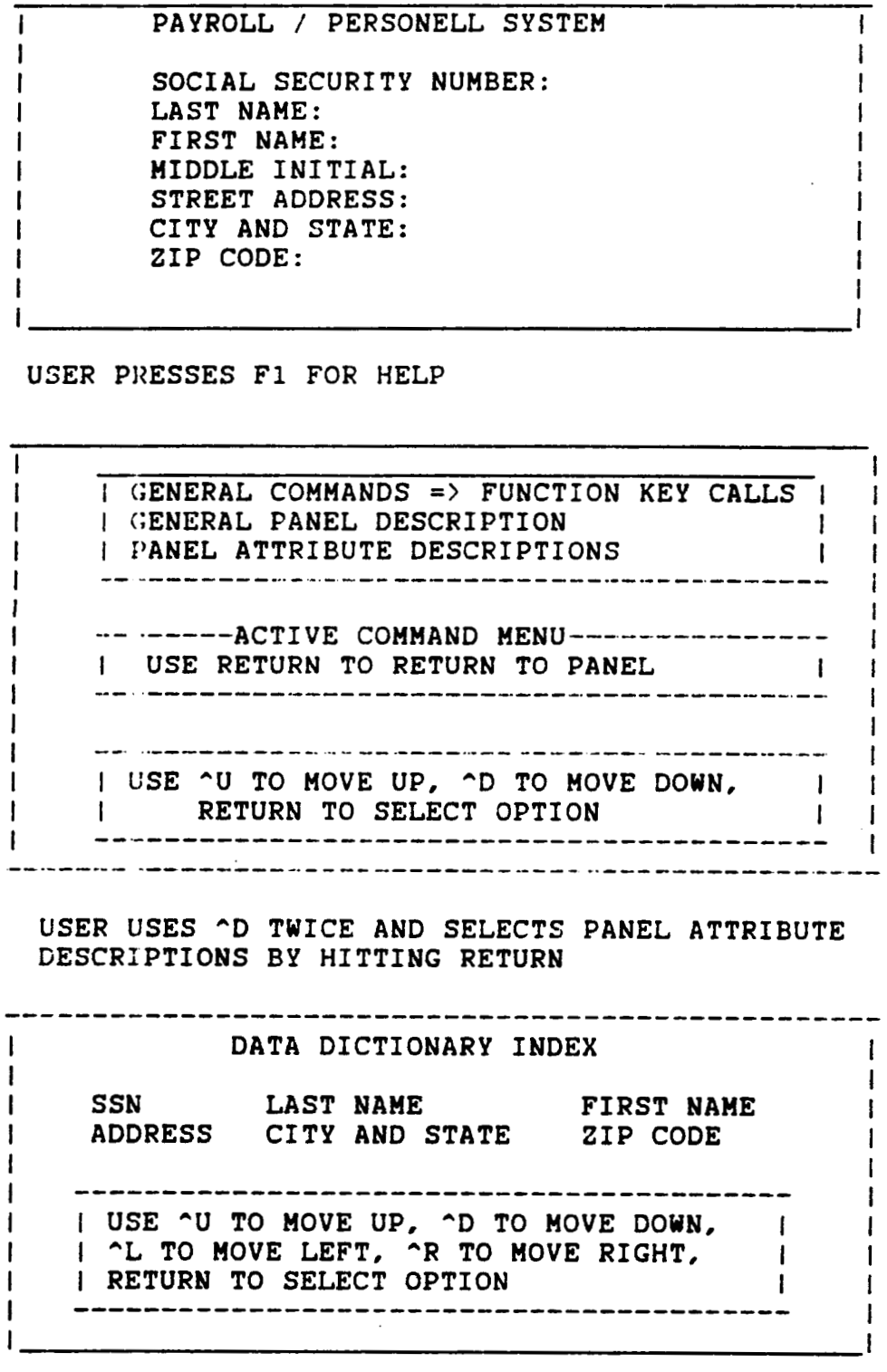
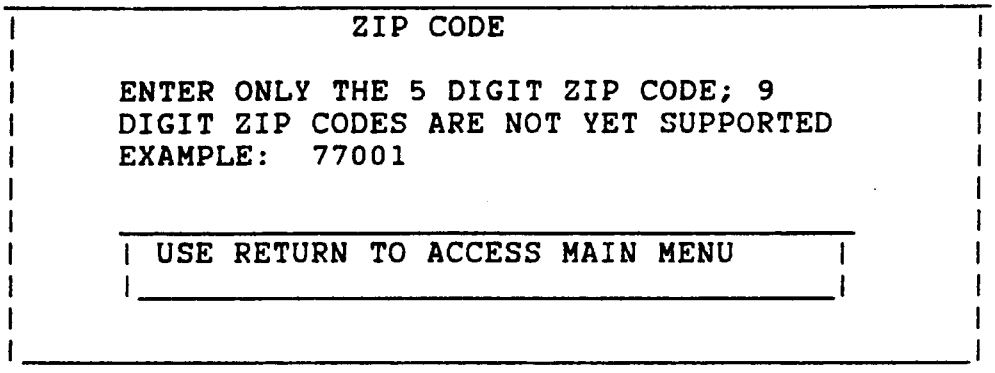


Figure 1. Typical session from user's point of view



RETURN GETS USER BACK TO MAIN MENU FROM
WHERE FURTHER HELP MAY BE OBTAINED OR
THE USER MAY EXIT BACK TO THE PROGRAM

Figure 1, continued.

Thus, the program had to be smart enough to know where the call was issued from, call the appropriate menu/manual, respond to the user's inputs and then return the user to the point from which the help call was made. We also decided that, in order to improve the user friendliness aspect of the program, the user should have little or no typing to do - i.e., the program should be able to respond to special purpose function key inputs.

Of course, the use of the help manual itself must be self-descriptive in order to relieve the user from having to turn to a manual on the on-line help manual.

Figure 1 shows what a typical session, from the end-users point of view, might look like.

3. Implementation Particulars

3.1 The Text File

The text file created by the implementor of a particular help facility has certain restrictions and rules. Figure 2 illustrates what the text file should "look" like.

The name of the help file corresponds to the name by which the help file is identified within the program. This name is not seen by the user, only by the program. This name should always be the first line of the text file.

Following the name of the help file comes the "name" of the introduction - i.e., a name with which or by which the user can identify the help package accessed. This name also appears on a line by itself. Following the introduction name is the textual description of the introduction itself.

Following the introduction, the name of each menu selection which will be made available to the user, along with the explanation which will be provided if the user selects that option, appears. Each name (including the introduction name) appears on a separate line and each description or explanation is terminated with a # terminator, also on a line of its own.

The current limits imposed on the names and descriptions in the text file are:

- the names are limited to 15 character in length,
- the textual descriptions for each menu option and for the introduction are limited to 24 lines of 80 characters each, and
- 65 such descriptions (including the introduction description) can exist.

These limits are, of course, program constants that can be changed, as necessary, to meet the requirements at hand.

3.2 The help file package

Once the text file has been created, the creator can incorporate his or her help facility into the general help package by calling a program called PROTOTYPE. The basic task of PROTOTYPE is to create a binary file containing the information of the text file.

When PROTOTYPE is called, the contents of a binary file called

```
Payroll/personel
Payroll system
This panel allows input of new or update of existing
personell records regarding payroll
#
SSN
The employee's social security number, in the following
format: 111 22 3333
#
Last name
Employee's last name, up to 20 characters in length.
Upper and/or lowercase letters may be used.
#
First name
Employee's first name, up to 20 character's in length.
Upper and/or lower case letters may be used.
Example: Employee's name is Marie Elizabeth Ogden;
         enter Marie as first name, even if employee goes
         by a different name.
#
```

Figure 2. Partial contents of a typical text file containing information to produce an on-line help manual

INDEX.BIN are loaded into memory. This binary file is strictly a set of names and associated indices (or read/write head positions). Figure 3 illustrates the contents of INDEX.BIN.

PROTOTYPE first reads the name of the help package from the text file. This name is stored in the next available position in an array of records which contains help panel names and pointers to their arrays of menu selection options, described below. The first position in the array contains a count of the total number of help files available; this number must be updated each time a new help facility is added to the system. The indices or pointers associated with each name are actually read/write head positions into INDEX.BIN itself where the set of menu selections are listed in an array associated with that particular help package.

After the help package name, in the text file, the introduction and menu selection names, along with their descriptions, are found. PROTOTYPE reads the name of the introduction or menu selection option and stores the name in the next available position in an array of menu selection names for that particular help facility. Once again, the first element of each of these arrays contains a count of the number of menu selection options (including the introduction) available through the package. (The actual position of the introduction name is always the second array position since it is always the first description to follow the help package name in the text file.) PROTOTYPE then reads the textual description associated with the last name read and when the terminator (#) is encountered, it performs a DIRECT_IO write of the description into a file called DIRECT.BIN. The read/write head position of the write is stored in INDEX.BIN along with the menu selection/introduction name with which it is associated.

When EOF is encountered in the text file, and no violations have occurred, PROTOTYPE performs a DIRECT_IO write of the updated version of INDEX.BIN. Any violation of syntax rules encountered in the text file during the above process causes an abort, with no updating of the INDEX.BIN binary file.

The second direct_io or binary file is the DIRECT.BIN mentioned above. It simply contains the textual descriptions, in binary form, which the help package will use.

Figure 4 illustrates the relationship between INDEX.BIN and DIRECT.BIN.

4. Discussion

When the user request on-line help he or she first gets a general menu where one of three general options can be selected. These are:

1- a set of generalized commands/keys which pertain to all help packages - e.g., how to save his/her work, how to exit to the system without saving the work, etc.
2- The introductory section to the help facility which contains a general description of the help package itself and of the program/package with which it is associated.
3- A listing of the menu selection options available to the user.

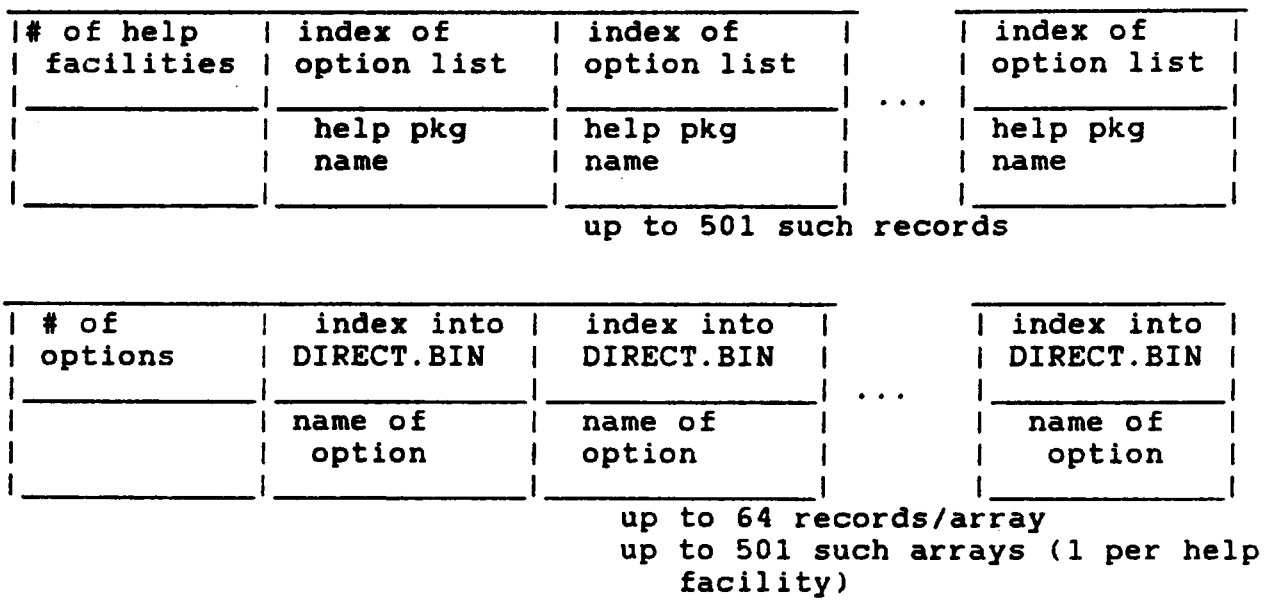


Figure 3. The basic structure of the direct_io file INDEX.BIN

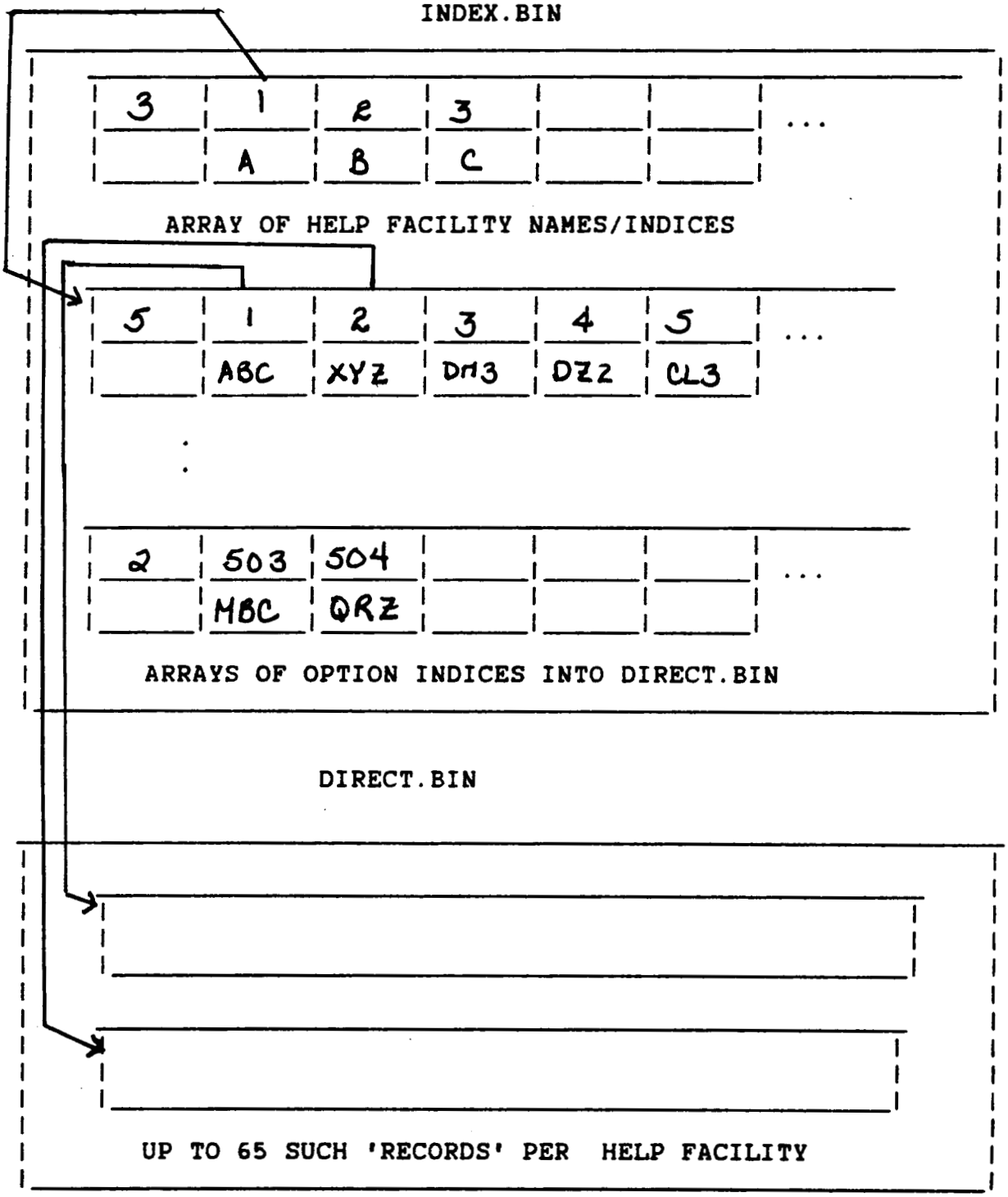


Figure 4. Relationship between INDEX.BIN and DIRECT.BIN

Each screen produced by a user selection contains a general set of directions for cursor movement, returning to previous screens, exiting, etc.

When the user selects option 1 above, PROTOTYPE is by-passed and the ON-LINE-HELP package calls a panel generation package to produce the "general help" screen, since the contents of this screen are the same regardless of where it was called from.

Options 2 and 3, on the other hand, use the information created by PROTOTYPE discussed above. When either of these is selected, INDEX.BIN is opened and the name of the help package associated with the program the user is in is searched for. When located, the pointers into DIRECT.BIN are made available through the array of pointers associated with that name. If option 2 was selected, the introduction section pointer (which, as mentioned earlier, is always the second record of the particular array) is used as the read/write head position into DIRECT.BIN where the textual description can be found.

If option 3 was selected by the user, a listing of menu options available is produced on the screen. This listing is available directly from INDEX.BIN since the names of the menu options (along with their indices into DIRECT.BIN) are stored therein. Since the menu selection items are listed in the order in which they appear in the INDEX.BIN array for that package, keeping track of the cursor movement, through simple addition and subtraction, also keeps track of the index into the array DIRECT.BIN for the element pointed to by the cursor. Thus, when the user does select a menu option, the index into DIRECT.BIN is directly available and the requested information can be displayed.

Figures 5 and 6 illustrate actual screens which the user might encounter; these are self-descriptive.

When the user exits the help mode, he or she is returned to the point from which the help call was made. This is made possible by saving the user's last used 'screen' in a buffer whose contents are re-displayed when the help facility is exited.

5. Summary and conclusion

As can be deduced from the above discussions, the idea behind the help facility is relatively simple. It is made unique by the fact that it is written in Ada and uses aspects of the language which make information retrieval rapid and simple. Specifically, the DIRECT_IO facility allows for random access into the help files. It is unnecessary to discuss the advantages of random access over sequential access.

The mere fact that the program is written in Ada implies a saving in terms of lines of code. This introduces the possiblity of eventually adapting the program to run at the micro-computer level, a major consideration in this day and age.

Additionally, since the program uses only standard Ada generics, it is portable to other systems. This is another aspect which must always be taken into consideration in writting any software package in the modern day world of computer programming.

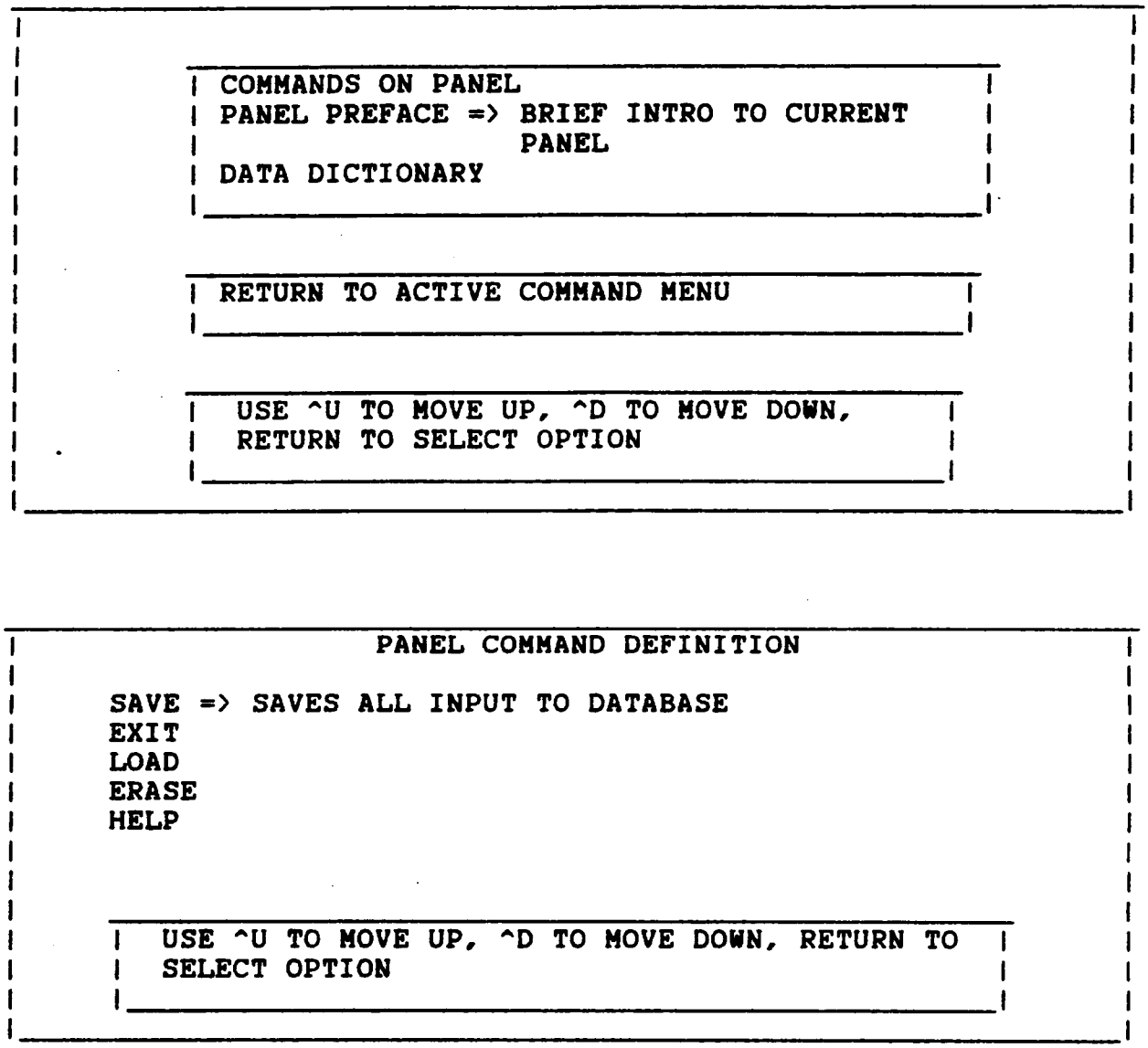

COMMANDS ON PANEL
PANEL PREFACE => BRIEF INTRO TO CURRENT PANEL
DATA DICTIONARY
RETURN TO ACTIVE COMMAND MENU
USE ^U TO MOVE UP, ^D TO MOVE DOWN, RETURN TO SELECT OPTION
PANEL COMMAND DEFINITION
SAVE => SAVES ALL INPUT TO DATABASE
EXIT
LOAD
ERASE
HELP
USE ^U TO MOVE UP, ^D TO MOVE DOWN, RETURN TO SELECT OPTION

Figure 5.

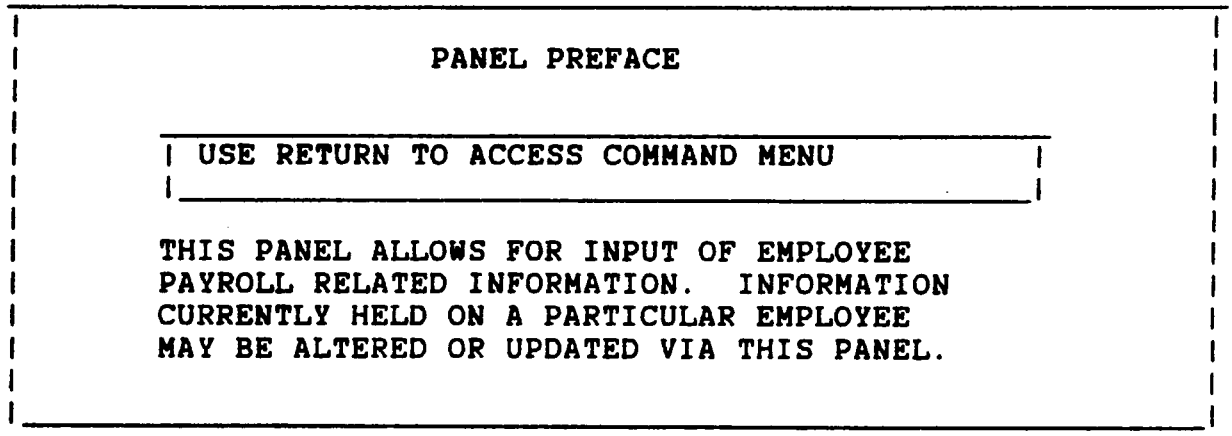


Figure 6.

*Graduate students at The University of Houston--Clear Lake, Houston, Texas

SESSION F.5

REUSABILITY PANEL

Panel Chair:

Delores S. Moorehead
Intermetrics
Houston, Texas

Panel Members:

Ron McCain
IBM Federal Systems Division
Houston, Texas

Ed Berard
EVB Software Engineering, Inc
Rockville, Maryland

Daniel McNichol
McDonnell-Douglas Astronautics Co.
St. Louis, Missouri

Rick Blumberg
Planning Research Corp.
McLean, Virginia

Norm Nise
California State Polytechnic University
and
Rockwell International
Downey, California

Elizabeth Wald
Naval Research Laboratory/STARS

SESSION F.6

DISTRIBUTED Ada PANEL

Panel Chair:

Roger Racine
The Charles Stark Draper Laboratory
Cambridge, Massachusetts

Panel Members:

Mike Kamrad
Honeywell Systems and Research Center
Minneapolis, Minnesota

Pat Rogers (Charles W. McKay)
University of Houston_Clear Lake
Houston, Texas

Trevor Mudge
Department of Electrical Engineering
University of Michigan
Ann Arbor, Michigan